particleswarm

Particle swarm optimization

Syntax

```
x = particleswarm(fun,nvars)
x = particleswarm(fun,nvars,lb,ub)
x = particleswarm(fun,nvars,lb,ub,options)
x = particleswarm(problem)
[x,fval,exitflag,output] = particleswarm( ___ )
```

Description

`x = particleswarm(fun,nvars)` attempts to find a vector `x` that achieves a local minimum of `fun`. `nvars` is the dimension (number of design variables) of `fun`. example

Note

Passing Extra Parameters (Optimization Toolbox) explains how to pass extra parameters to the objective function, if necessary.

`x = particleswarm(fun,nvars,lb,ub)` defines a set of lower and upper bounds on the design variables, `x`, so that a solution is found in the range `lb ≤ x ≤ ub`. example

`x = particleswarm(fun,nvars,lb,ub,options)` minimizes with the default optimization parameters replaced by values in `options`. Set `lb = []` and `ub = []` if no bounds exist. example

`x = particleswarm(problem)` finds the minimum for `problem`, where `problem` is a structure.

`[x,fval,exitflag,output] = particleswarm( ___ )`, for any input arguments described above, returns: example

- A scalar `fval`, which is the objective function value `fun(x)`
- A value `exitflag` describing the exit condition
- A structure `output` containing information about the optimization process

Examples

collapse all

Minimize a Simple Function

Minimize a simple function of two variables.

Define the objective function.

```
fun = @(x)x(1)*exp(-norm(x)^2);
```

Call `particleswarm` to minimize the function.

```
rng default  % For reproducibility
nvars = 2;
x = particleswarm(fun,nvars)
```

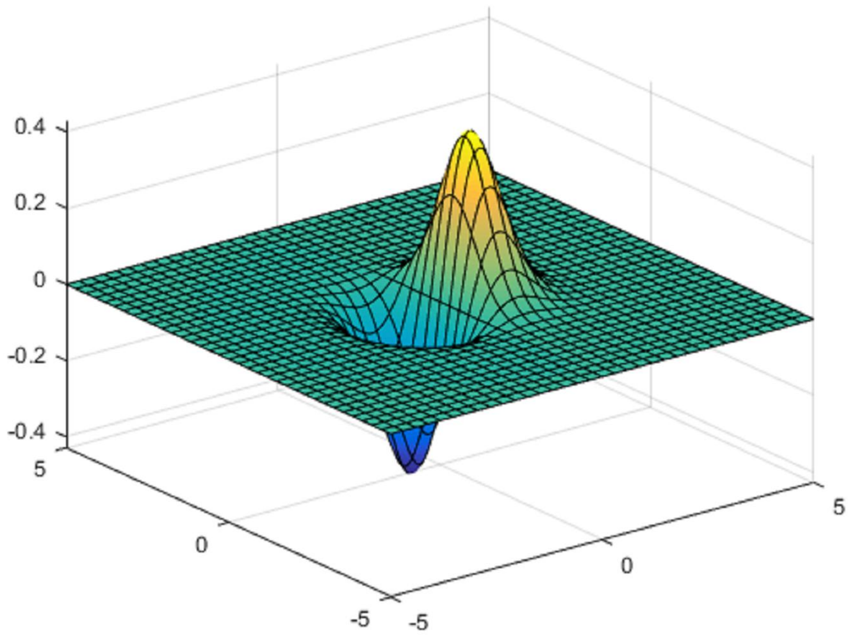
```
Optimization ended: relative change in the objective value
over the last OPTIONS.MaxStallIterations iterations is less than OPTIONS.FunctionTolerance.

x =

  629.4474  311.4814
```

This solution is far from the true minimum, as you see in a function plot.

```
fsurf(@(x,y)x.*exp(-(x.^2+y.^2)))
```



Usually, it is best to set bounds. See Minimize a Simple Function with Bounds.

Minimize a Simple Function with Bounds

Minimize a simple function of two variables with bound constraints.

View MATLAB Command

Define the objective function.

```
fun = @(x)x(1)*exp(-norm(x)^2);
```

Set bounds on the variables.

```
lb = [-10,-15];
ub = [15,20];
```

Call `particleswarm` to minimize the function.

```
rng default  % For reproducibility
nvars = 2;
x = particleswarm(fun,nvars,lb,ub)
```

```
Optimization ended: relative change in the objective value
over the last OPTIONS.MaxStallIterations iterations is less than OPTIONS.FunctionTolerance.
```

```
x = 1×2

   -0.7071   -0.0000
```

Minimize Using Nondefault Options

Use a larger population and a hybrid function to try to get a better solution.

View MATLAB Command

Specify the objective function and bounds.

```
fun = @(x)x(1)*exp(-norm(x)^2);
lb = [-10,-15];
ub = [15,20];
```

Specify the options.

```
options = optimoptions('particleswarm','SwarmSize',100,'HybridFcn',@fmincon);
```

Call `particleswarm` to minimize the function.

```
rng default  % For reproducibility
nvars = 2;
x = particleswarm(fun,nvars,lb,ub,options)
```

```
Optimization ended: relative change in the objective value
over the last OPTIONS.MaxStallIterations iterations is less than OPTIONS.FunctionTolerance.
x = 1×2

   -0.7071   -0.0000
```

Examine the Solution Process

Return the optional output arguments to examine the solution process in more detail.

View MATLAB Command

Define the problem.

```
fun = @(x)x(1)*exp(-norm(x)^2);
lb = [-10,-15];
ub = [15,20];
options = optimoptions('particleswarm','SwarmSize',50,'HybridFcn',@fmincon);
```

Call `particleswarm` with all outputs to minimize the function and get information about the solution process.

```
rng default  % For reproducibility
nvars = 2;
[x,fval,exitflag,output] = particleswarm(fun,nvars,lb,ub,options)
```

```
Optimization ended: relative change in the objective value
over the last OPTIONS.MaxStallIterations iterations is less than OPTIONS.FunctionTolerance.
```

```
x = 1×2

   -0.7071   -0.0000

fval = -0.4289
exitflag = 1
output = struct with fields:
      rngstate: [1x1 struct]
    iterations: 43
     funccount: 2203
       message: 'Optimization ended: relative change in the objective value ...'
```

Input Arguments

collapse all

fun — Objective function

function handle | function name

Objective function, specified as a function handle or function name. Write the objective function to accept a row vector of length nvars and return a scalar value.

When the 'UseVectorized' option is true, write fun to accept a pop-by-nvars matrix, where pop is the current population size. In this case, fun returns a vector the same length as pop containing the fitness function values. Ensure that fun does not assume any particular size for pop, since particleswarm can pass a single member of a population even in a vectorized calculation.

Example: fun = @(x)(x-[4,2]).^2

Data Types: char | function_handle | string

nvars — Number of variables

positive integer

Number of variables, specified as a positive integer. The solver passes row vectors of length nvars to fun.

Example: 4

Data Types: double

lb — Lower bounds

[ ] (default) | real vector or array

Lower bounds, specified as a real vector or array of doubles. lb represents the lower bounds element-wise in lb ≤ x ≤ ub.

Internally, particleswarm converts an array lb to the vector lb(:).

Example: lb = [0;-Inf;4] means x(1) ≥ 0, x(3) ≥ 4.

Data Types: double

ub — Upper bounds

[ ] (default) | real vector or array

Upper bounds, specified as a real vector or array of doubles. `ub` represents the upper bounds element-wise in `lb ≤ x ≤ ub`.

Internally, `particleswarm` converts an array `ub` to the vector `ub(:)`.

Example: `ub = [Inf;4;10]` means `x(2) ≤ 4`, `x(3) ≤ 10`.

Data Types: `double`

`options` — Options for `particleswarm`
options created using `optimoptions`

Options for `particleswarm`, specified as the output of the `optimoptions` function.

Some options are absent from the `optimoptions` display. These options are listed in italics. For details, see View Options (Optimization Toolbox).

`CreationFcn`	Function that creates the initial swarm. Specify as `'pswcreationuniform'` or a function handle. Default is `'pswcreationuniform'`. See Swarm Creation.
`Display`	Level of display returned to the command line. • `'off'` or `'none'` displays no output. • `'final'` displays just the final output (default). • `'iter'` gives iterative display.
DisplayInterval	Interval for iterative display. The iterative display prints one line for every `DisplayInterval` iterations. Default is `1`.
`FunctionTolerance`	Nonnegative scalar with default `1e-6`. Iterations end when the relative change in best objective function value over the last `MaxStallIterations` iterations is less than `options.FunctionTolerance`.
FunValCheck	Check whether objective function and constraints values are valid. `'on'` displays an error when the objective function or constraints return a value that is complex, `Inf`, or `NaN`. The default, `'off'`, displays no error.
`HybridFcn`	Function that continues the optimization after `particleswarm` terminates. Specify as a name or a function handle. Possible values: • `'fmincon'` • `'fminsearch'` • `'fminunc'` • `'patternsearch'` Can also be a cell array specifying the hybrid function and its options, such as `{@fmincon,fminconopts}`. Default is `[]`. See Hybrid Function. See When to Use a Hybrid Function.
`InertiaRange`	Two-element real vector with same sign values in increasing order. Gives the lower and upper bound of the adaptive inertia. To obtain a constant (nonadaptive) inertia, set both elements of `InertiaRange` to the same value. Default is `[0.1,1.1]`. See Particle Swarm Optimization Algorithm.

`InitialSwarmMatrix`	Initial population or partial population of particles. `M`-by-`nvars` matrix, where each row represents one particle. If `M < SwarmSize`, then `particleswarm` creates more particles so that the total number is `SwarmSize`. If `M > SwarmSize`, then `particleswarm` uses the first `SwarmSize` rows.
`InitialSwarmSpan`	Initial range of particle positions that `@pswcreationuniform` creates. Can be a positive scalar or a vector with `nvars` elements, where `nvars` is the number of variables. The range for any particle component is `-InitialSwarmSpan/2,InitialSwarmSpan/2`, shifted and scaled if necessary to match any bounds. Default is `2000`. `InitialSwarmSpan` also affects the range of initial particle velocities. See Initialization.
`MaxIterations`	Maximum number of iterations `particleswarm` takes. Default is `200*nvars`, where `nvars` is the number of variables.
`MaxStallIterations`	Positive integer with default `20`. Iterations end when the relative change in best objective function value over the last `MaxStallIterations` iterations is less than `options.FunctionTolerance`.
`MaxStallTime`	Maximum number of seconds without an improvement in the best known objective function value. Positive scalar with default `Inf`.
`MaxTime`	Maximum time in seconds that `particleswarm` runs. Default is `Inf`.
`MinNeighborsFraction`	Minimum adaptive neighborhood size, a scalar from `0` to `1`. Default is `0.25`. See Particle Swarm Optimization Algorithm.
`ObjectiveLimit`	Minimum objective value, a stopping criterion. Scalar, with default `-Inf`.
`OutputFcn`	Function handle or cell array of function handles. Output functions can read iterative data, and stop the solver. Default is `[]`. See Output Function and Plot Function.
`PlotFcn`	Function name, function handle, or cell array of function handles. For custom plot functions, pass function handles. Plot functions can read iterative data, plot each iteration, and stop the solver. Default is `[]`. Available built-in plot function: `'pswplotbestf'`. See Output Function and Plot Function.
`SelfAdjustmentWeight`	Weighting of each particle’ s best position when adjusting velocity. Finite scalar with default `1.49`. See Particle Swarm Optimization Algorithm.
`SocialAdjustmentWeight`	Weighting of the neighborhood’ s best position when adjusting velocity. Finite scalar with default `1.49`. See Particle Swarm Optimization Algorithm.
`SwarmSize`	Number of particles in the swarm, an integer greater than `1`. Default is `min(100,10*nvars)`, where `nvars` is the number of variables.
`UseParallel`	Compute objective function in parallel when `true`. Default is `false`. See Parallel or Vectorized Function Evaluation.
`UseVectorized`	Compute objective function in vectorized fashion when `true`. Default is `false`. See Parallel or Vectorized Function Evaluation.

`problem` — Optimization problem
structure

Optimization problem, specified as a structure with the following fields.

`solver`	`'particleswarm'`

`objective`	Function handle to the objective function, or name of the objective function.
`nvars`	Number of variables in problem.
`lb`	Vector or array of lower bounds.
`ub`	Vector or array of upper bounds.
`options`	Options created by `optimoptions`.
`rngstate`	Optional state of the random number generator at the beginning of the solution process.

Data Types: `struct`

Output Arguments

collapse all

x — Solution
real vector

Solution, returned as a real vector that minimizes the objective function subject to any bound constraints.

`fval` — Objective value
real scalar

Objective value, returned as the real scalar `fun(x)`.

`exitflag` — Algorithm stopping condition
integer

Algorithm stopping condition, returned as an integer identifying the reason the algorithm stopped. The following lists the values of `exitflag` and the corresponding reasons `particleswarm` stopped.

`1`	Relative change in the objective value over the last `options.MaxStallIterations` iterations is less than `options.FunctionTolerance`.
`0`	Number of iterations exceeded `options.MaxIterations`.
`-1`	Iterations stopped by output function or plot function.
`-2`	Bounds are inconsistent: for some `i`, `lb(i) > ub(i)`.
`-3`	Best objective function value is at or below `options.ObjectiveLimit`.
`-4`	Best objective function value did not change within `options.MaxStallTime` seconds.
`-5`	Run time exceeded `options.MaxTime` seconds.

`output` — Solution process summary
structure

Solution process summary, returned as a structure containing information about the optimization process.

`iterations`	Number of solver iterations
`funccount`	Number of objective function evaluations.
`message`	Reason the algorithm stopped.
`rngstate`	State of the default random number generator just before the algorithm started.

Algorithms

For a description of the particle swarm optimization algorithm, see Particle Swarm Optimization Algorithm.

Extended Capabilities

Automatic Parallel Support

Accelerate code by automatically running computation in parallel using Parallel Computing Toolbox™.

See Also

`ga` | `patternsearch`

Topics

Optimize Using Particle Swarm

Particle Swarm Output Function

What Is Particle Swarm Optimization?

Optimization Problem Setup

Introduced in R2014b

Particle Swarm Optimization Algorithm

Algorithm Outline

particleswarm is based on the algorithm described in Kennedy and Eberhart [1], using modifications suggested in Mezura-Montes and Coello Coello [2] and in Pedersen [3].

The particle swarm algorithm begins by creating the initial particles, and assigning them initial velocities.

It evaluates the objective function at each particle location, and determines the best (lowest) function value and the best location.

It chooses new velocities, based on the current velocity, the particles' individual best locations, and the best locations of their neighbors.

It then iteratively updates the particle locations (the new location is the old one plus the velocity, modified to keep particles within bounds), velocities, and neighbors.

Iterations proceed until the algorithm reaches a stopping criterion.

Here are the details of the steps.

Initialization

By default, particleswarm creates particles at random uniformly within bounds. If there is an unbounded component, particleswarm creates particles with a random uniform distribution from –1000 to 1000. If you have only one bound, particleswarm shifts the creation to have the bound as an endpoint, and a creation interval 2000 wide. Particle i has position x(i), which is a row vector with nvars elements. Control the span of the initial swarm using the InitialSwarmSpan option.

Similarly, particleswarm creates initial particle velocities v at random uniformly within the range [-r,r], where r is the vector of initial *ranges*. The range of component k is min(ub(k) - lb(k),InitialSwarmSpan(k)).

particleswarm evaluates the objective function at all particles. It records the current position p(i) of each particle i. In subsequent iterations, p(i) will be the location of the best objective function that particle i has found. And b is the best over all particles: b = min(fun(p(i))). d is the location such that b = fun(d).

particleswarm initializes the neighborhood size N to minNeighborhoodSize = max(2,floor(SwarmSize*MinNeighborsFraction)).

particleswarm initializes the inertia W = max(InertiaRange), or if InertiaRange is negative, it sets W = min(InertiaRange).

particleswarm initializes the stall counter c = 0.

For convenience of notation, set the variable y1 = SelfAdjustmentWeight, and y2 = SocialAdjustmentWeight, where SelfAdjustmentWeight and SocialAdjustmentWeight are options.

Iteration Steps

The algorithm updates the swarm as follows. For particle `i`, which is at position `x(i)`:

1. Choose a random subset `S` of `N` particles other than `i`.
2. Find `fbest(S)`, the best objective function among the neighbors, and `g(S)`, the position of the neighbor with the best objective function.
3. For `u1` and `u2` uniformly (0,1) distributed random vectors of length `nvars`, update the velocity

   ```
   v = W*v + y1*u1.*(p-x) + y2*u2.*(g-x).
   ```

   This update uses a weighted sum of:

   - The previous velocity `v`
   - The difference between the current position and the best position the particle has seen `p-x`
   - The difference between the current position and the best position in the current neighborhood `g-x`

4. Update the position `x = x + v`.
5. Enforce the bounds. If any component of `x` is outside a bound, set it equal to that bound. For those components that were just set to a bound, if the velocity `v` of that component points outside the bound, set that velocity component to zero.
6. Evaluate the objective function `f = fun(x)`.
7. If `f < fun(p)`, then set `p = x`. This step ensures `p` has the best position the particle has seen.
8. The next steps of the algorithm apply to parameters of the entire swarm, not the individual particles. Consider the smallest `f = min(f(j))` among the particles `j` in the swarm.

   If `f < b`, then set `b = f` and `d = x`. This step ensures `b` has the best objective function in the swarm, and `d` has the best location.
9. If, in the previous step, the best function value was lowered, then set `flag = true`. Otherwise, `flag = false`. The value of `flag` is used in the next step.
10. Update the neighborhood. If `flag = true`:

    a. Set `c = max(0,c-1)`.

    b. Set `N` to `minNeighborhoodSize`.

    c. If `c < 2`, then set `W = 2*W`.

    d. If `c > 5`, then set `W = W/2`.

    e. Ensure that `W` is in the bounds of the `InertiaRange` option.

    If `flag = false`:

    a. Set `c = c+1`.

    b. Set `N = min(N + minNeighborhoodSize,SwarmSize)`.

Stopping Criteria

`particleswarm` iterates until it reaches a stopping criterion.

Stopping Option	Stopping Test	Exit Flag
`MaxStallIterations` and `FunctionTolerance`	Relative change in the best objective function value `g` over the last `MaxStallIterations` iterations is less than `FunctionTolerance`.	`1`
`MaxIterations`	Number of iterations reaches `MaxIterations`.	`0`
`OutputFcn` or `PlotFcn`	`OutputFcn` or `PlotFcn` can halt the iterations.	`-1`
`ObjectiveLimit`	Best objective function value `g` is less than or equal to `ObjectiveLimit`.	`-3`
`MaxStallTime`	Best objective function value `g` did not change in the last `MaxStallTime` seconds.	`-4`
`MaxTime`	Function run time exceeds `MaxTime` seconds.	`-5`

If `particleswarm` stops with exit flag `1`, it optionally calls a hybrid function after it exits.

References

[1] Kennedy, J., and R. Eberhart. "Particle Swarm Optimization." *Proceedings of the IEEE International Conference on Neural Networks.* Perth, Australia, 1995, pp. 1942–1945.

[2] Mezura-Montes, E., and C. A. Coello Coello. "Constraint-handling in nature-inspired numerical optimization: Past, present and future." *Swarm and Evolutionary Computation.* 2011, pp. 173–194.

[3] Pedersen, M. E. "Good Parameters for Particle Swarm Optimization." Luxembourg: Hvass Laboratories, 2010.

Related Topics

- What Is Particle Swarm Optimization?
- Optimize Using Particle Swarm